Учреждение образования

«Гродненский государственный политехнический колледж»

Специальность: 2 – 40 01 01 «Программное обеспечение информационных

технологий»

Специализация: 2 – 40 01 01 33 «Компьютерная графика»

Предмет: «Технология разработки программного обеспечения»

Группа: ПЗТ – 41

**ТЕХНО-РАБОЧИЙ ПРОЕКТ**

Техническое задание на разработку чат-бота «Кабинет Школьного Психолога»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

Разработал Н. С. Шаматульский

Руководитель проекта О. И. Кизер

2024

**Введение**

На учебную практику по «Разработке и сопровождению программного обеспечения» выбрана тема разработки чат-бота «Кабинет Школьного Психолога».

В данном разделе представлено краткое описание темы разработки, связанной с созданием телеграм-бота для школьных психологов.

Основной задачей данного проекта является разработка программного продукта, который служит инструментом для оказания психологической поддержки учащимся, а также для упрощения работы социальных психологов. Чат-бот предоставляет возможность анонимного общения с психологом, получения рекомендаций по управлению эмоциональным состоянием, а также получения полезных статей и новостей. Чат-бот позволяет педагогам-психологам создавать собственные анкеты для проведения опросов, избавляя от необходимости использовать бумажные варианты, что значительно упрощает задачу сбора и проведения анкет, а также обеспечивает удобный и быстрый доступ к полученным данным. В условиях современного образовательного процесса, где психическое здоровье становится все более актуальной темой, создание такого бота представляет собой важный шаг к обеспечению доступной и своевременной помощи.

Актуальность поставленной задачи обусловлена необходимостью повышения уровня психологической поддержки в образовательных учреждениях. Учащиеся часто сталкиваются с различными стрессовыми ситуациями, такими как экзамены, конфликты с одноклассниками, проблемы в семье и другие факторы, способствующие возникновению тревожности и депрессии. Наличие доступного инструмента для получения помощи может существенно улучшить их эмоциональное состояние и общее благополучие. Назначение данного проекта заключается в создании удобного и интуитивно понятного интерфейса, который позволяет пользователям легко взаимодействовать с ботом и получать необходимую информацию.

Пояснительная записка состоит из нескольких разделов, каждый из которых охватывает ключевые аспекты разработки.

В первом разделе «Анализ задачи» формулируется постановка задачи проекта, цель разработки, а также определяются требования к функционалу бота.

В разделе «Проектирование задачи» рассматриваются основные аспекты разработки программного продукта. Здесь можно узнать об организации данных в контексте среды разработки. В данном разделе четко описывается пользовательский интерфейс, составляются алгоритмы процесса обработки информации.

В третьем разделе «Реализация задачи» описываются все элементы и объекты, которые используются при реализации данного чат-бота. В этом разделе четко описываются функции пользователя и их структура.

Четвертый раздел называется «Тестирование». В нем проводится тестирование чат-бота, охватывающее каждый пункт меню и все доступные операции. Моделируются все возможные действия пользователя при взаимодействии с ботом.

Последний раздел – «Руководство пользователя» включает в себя краткое изложение цели разработки проекта и предоставляет инструкции по использованию программного продукта.

В «Заключении» представляется сжатое изложение задачи, результаты выполненной работы, а также описание применяемых методов и инструментов.

В «Списке использованных источников» приводится список используемых при разработке источников с информацией.

В приложении А приведена диаграмма вариантов использования.

В приложении Б приведена диаграмма Ганта.

В приложении В приведена структура чат-бота.

В приложении Г приведена модель данных.

В приложении Д приведена диаграмма последовательности.

В приложении Е приведена диаграмма деятельности.

В приложении Ж приведена диаграмма классов.

В приложении К приведен пользовательский интерфейс программы.

В приложении Л приведены тест-кейсы.

В приложени П приведен листинг программы.

**1 Анализ задачи**

**1.1 Постановка задачи**

**1.1.1 Организационно-экономическая сущность задачи**

Наименование задачи: чат-бот «Кабинет Школьного Психолога»

Цель разработки: предоставление психологической поддержки ученикам и облегчение доступа к информации о психическом здоровье. Бот упростит сбор результатов анкет для психологов и позволит учащимся проходить опросы на телефоне, исключая бумажные формы.

Назначение: бот будет предоставлять комплексную психологическую поддержку, помогая ученикам понять свои чувства и получить актуальную информацию о психическом здоровье. Учащиеся смогут проходить опросы и тесты для оценки своего состояния, а педагоги-психологи — создавать анкеты для сбора данных.

Периодичность использования: от ежедневных до ежемесячных обращений, в любое время по мере необходимости.

Источники данных: бот будет использовать статьи, новости, научные исследования и рекомендации психологов, а также интегрировать материалы из проверенных онлайн-ресурсов и видеоматериалы по психическому здоровью. Психологи смогут публиковать статьи и новости, а администраторы — редактировать контент.

Информационная связь: бот будет связан с социальными сетями и новостными порталами для предоставления актуальной информации.

Анализ аналогичных приложений: чат-бот «Помощь» в Бресте предлагает анонимную поддержку, но наш бот «Кабинет Школьного Психолога» предоставляет более широкий спектр возможностей, включая доступ к образовательным материалам и тестам. Он создает платформу для обучения и профессионального взаимодействия, удовлетворяя потребности учащихся в психоэмоциональном благополучии.

**1.1.2 Функциональные требования**

Будущий программный продукт должен выполнять ряд функций для различных пользователей. Учащиеся школы смогут авторизоваться, получать анонимные консультации у педагога-психолога, иметь доступ к статьям и ресурсам, а также проходить анкеты, психологические тесты и викторины. Школьные психологи смогут создавать авторские анкеты, поддерживать анонимные консультации и получать доступ к статьям. Администраторы будут иметь возможность блокировать и разблокировать пользователей, а также просматривать список заблокированных пользователей.

Входные данные пользователей будут вноситься в базу данных, организованные в файлы формата СУБД MySQL.

Формы представления документов по каждой позиции перечня включают данные пользователей, которые представлены в виде таблицы в MySQL, анкеты и тесты, оформленные как сообщения в Telegram, а также ресурсы и материалы, также представленные в виде сообщений в Telegram.

Перечень результатной информации включает в себя: статистические данные о пользователях, такие как возраст и класс; информационные материалы по психологии, включая статьи, новости, советы и видеоматериалы; результаты викторин, тестов и анкет, например, тесты на определение уровня стресса и самооценки; а также анонимные обращения пользователей.

Формы представления информации включают печатную сводку, которая содержит отчеты о проведенных опросах и результатах тестов в формате MySQL (.db). Данные также могут быть экспортированы на машинный носитель в Excel для дальнейшего анализа. Кроме того, информация представляется через интерфейс бота в виде текстовых сообщений и кнопок для взаимодействия. Периодичность и сроки представления информации зависят от запросов пользователей, что позволяет получать данные по конкретным запросам, а также включает сводки с результатами опросов, анкет и тестов.

Результатная информация предназначена для различных пользователей, включая подразделения, такие как психологическая служба школы и администрация школы. Среди персонала, который также получает доступ к этой информации, находятся школьные психологи и социальные работники.

В рамках описания условно-постоянной информации используются различные классификаторы, справочники и таблицы. Классификаторы, например, охватывают темы статей, связанные с психическим здоровьем, такие как стресс, тревожность, депрессия и самооценка, что позволяет пользователям легко находить нужную информацию. Также имеется классификатор типов тестов и опросов, который включает разнообразные инструменты, такие как тесты на уровень стресса и самооценку, что помогает пользователям выбирать подходящие методы для самооценки.

Справочники содержат информацию о ресурсах поддержки, включая службы помощи и горячие линии, что обеспечивает школьникам быстрый доступ к необходимым контактам. В дополнение к этому, результаты тестов представлены в сводной таблице, где указаны средние значения и распределение ответов, что позволяет анализировать собственные результаты и сравнивать их с общими показателями.

Информация представляется в различных форматах, включая текстовые сообщения внутри бота, что обеспечивает удобный доступ к данным. Для более подробных материалов, таких как справочники и таблицы, пользователи могут скачать документы в формате PDF или Word. Кроме того, навигационные элементы, такие как кнопки и меню, упрощают поиск нужных классификаторов и ресурсов, что делает взаимодействие с ботом более эффективным и комфортным.

Исходная информация предназначена для различных пользователей, включая учащихся школы, психологическую службу и администрацию школы.

**1.1.3 Нефункциональные (эксплуатационные) требования**

Будущий программный продукт будет выполнять функции для различных пользователей. Учащиеся смогут авторизоваться, получать анонимные консультации у педагога-психолога, доступ к статьям и ресурсам, а также проходить анкеты и тесты.

Школьные психологи смогут создавать авторские анкеты, поддерживать анонимные консультации и получать доступ к статьям. Администраторы смогут блокировать и разблокировать пользователей, а также просматривать список заблокированных.

Данные пользователей будут храниться в базе данных MySQL. Формы представления информации включают таблицы в MySQL, анкеты и тесты в Telegram, а также ресурсы и материалы в виде сообщений.

Результаты информации будут включать статистику пользователей (возраст, класс), материалы по психологии (статьи, советы), результаты тестов и анонимные обращения. Информация будет представлена в виде печатных сводок, экспортируемых в Excel, и через интерфейс бота.

Результаты информации предназначены для психологической службы и администрации школы, а также для школьных психологов и социальных работников.

В системе будут использоваться классификаторы и справочники. Классификаторы помогут находить статьи по темам, связанным с психическим здоровьем, а справочники предоставят информацию о службах поддержки.

Результаты тестов будут представлены в сводной таблице для анализа. Информация будет доступна в текстовых сообщениях бота, а более подробные материалы можно будет скачать в формате PDF или Word. Навигационные элементы упростят поиск ресурсов и классификаторов.

Исходная информация будет предназначена для учащихся, психологической службы и администрации школы.

**1.2 Диаграмма вариантов использования**

Диаграмма вариантов использования – это диаграмма, описывающая какой функционал разрабатываемой программной системы доступен каждой группе пользователей. В диаграмме данного сайта представлено всего 3 вида актеров: пользователь (ученику школы), школьный психолог, администратор. Функционал для каждого из них подробно описан в диаграмме, представленной в Приложении А.

**1.3 Выбор стратегии разработки и модели жизненного цикла**

Выбор стратегии разработки и модели жизненного цикла программного обеспечения — это ключевые этапы в процессе создания успешного продукта. Эти решения определяют не только подход к разработке, но и взаимодействие команды, управление проектом, а также конечное качество и функциональность создаваемого решения.

Правильный выбор стратегии и модели жизненного цикла позволяет минимизировать риски, оптимизировать ресурсы и обеспечить соответствие конечного продукта требованиям пользователей и бизнес-целям.

Ознакомимся с таблицами по выбору модели жизненного цикла на основе разных характеристик, которые представлены в таблицах 1-4.

Таблица 1 – Выбор модели жизненного цикла на основе характеристик требований

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории требований | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1 | Являются ли требования к проекту легко определимыми и реализуемыми? | Да | Да | Да | Нет | Нет | Нет |
| 2 | Могут ли требования быть сформулированы в начале ЖД? | Да | Да | Да | Да | Нет | Нет |
| 3 | Часто ли будут изменяться требования на протяжении ЖЦ? | Нет | Нет | Нет | Нет | Да | Да |

Продолжение таблицы 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | Нужно ли демонстрировать требования с целью их определения? | Нет | Нет | Да | Нет | Да | Да |
| 5 | Требуется ли проверка концепции программного средства или системы? | Нет | Нет | Да | Нет | Да | Да |
| 6 | Будут ли требования изменяться или уточняться с ростом сложности системы (программного средства) в ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7 | Нужно ли реализовать основные требования на ранних этапах разработки? | Нет | Нет | Да | Да | Да | Да |

Вычисления: 2 за каскадную, 2 за V-образную, 4 за RAD, 3 за инкрементную, 4 за быстрого прототипирования и 4 за эволюционную.

Итог: на основе результатов заполнения таблицы 1 подходящей является прототипированная модель, эволюционная модель и RAD.

Таблица 2 **–** Выбор модели жизненного цикла на основе характеристик команды разработчиков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории команды разработчиков  проекта | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Являются ли проблемы предметной области проекта новыми для большинства разработчиков? | Нет | Нет | Нет | Нет | Да | Да |

Продолжение таблицы 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2. | Являются ли инструментальные средства, используемые в проекте, новыми для большинства разработчиков? | Да | Да | Нет | Нет | Нет | Да |
| 3. | Изменяются ли роли участников проекта на протяжении ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Является ли структура процесса разработки более значимой для разработчиков, чем гибкость? | Да | Да | Нет | Да | Нет | Нет |
| 5. | Важна ли легкость распределения человеческих ресурсов проекта? | Да | Да | Да | Да | Нет | Нет |
| 6. | Приемлет ли команда разработчиков оценки, проверки, стадии разработки? | Да | Да | Нет | Да | Да | Да |

Вычисления: 2 за каскадную, 2 за V-образную, 1 за RAD, 2 за инкрементную, 1 за быстрого прототипирования и 1 за эволюционную.

Итог: на основании результатов заполнения таблицы 2 подходящими являются каскадная V-образная и инкрементная модели.

Таблица 3 **–** Выбор модели жизненного цикла на основе характеристик коллектива пользователей

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории коллектива пользователей | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Будет ли присутствие пользователей ограничено в ЖЦ разработки? | Да | Да | Нет | Да | Нет | Да |
| 2. | Будут ли пользователи оценивать текущее состояние программного продукта (системы) в процессе разработки? | Нет | Нет | Нет | Да | Да | Да |
| 3. | Будут ли пользователи вовлечены во все фазы ЖЦ разработки? | Нет | Нет | Да | Нет | Да | Нет |

Продолжение таблицы 3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 4. | Будет ли заказчик отслеживать ход выполнения проекта? | Нет | Нет | Нет | Нет | Да | Да |

Вычисления: 0 за каскадную, 0 за V-образную, 0 за RAD, 0 за инкрементную, 0 за быстрого прототипирования и 0 за эволюционную.

Итог: на основании результатов заполнения таблицы 3 подходящими являются каскадная V-образная и инкрементная модели.

Таблица 4 **–** Выбор модели жизненного цикла на основе характеристик типа проектов и рисков

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № критерия | Критерии категории типов проекта и рисков | Каскадная | V-образная | RAD | Инкрементная | Быстрого прототипирования | Эволюционная |
| 1. | Разрабатывается ли в проекте продукт нового для организации направления? | Нет | Нет | Нет | Да | Да | Да |
| 2. | Будет ли проект являться расширением существующей системы? | Да | Да | Да | Да | Нет | Нет |
| 3. | Будет ли проект крупно- или среднемасштабным? | Нет | Нет | Нет | Да | Да | Да |
| 4. | Ожидается ли длительная эксплуатация продукта? | Да | Да | Нет | Да | Нет | Да |
| 5. | Необходим ли высокий уровень надежности продукта проекта? | Нет | Да | Нет | Да | Нет | Да |
| 6. | Предполагается ли эволюция продукта проекта в течение ЖЦ? | Нет | Нет | Нет | Да | Да | Да |
| 7. | Велика ли вероятность изменения системы (продукта) на этапе сопровождения? | Нет | Нет | Нет | Да | Да | Да |
| 8. | Является ли график сжатым? | Нет | Нет | Да | Да | Да | Да |
| 9. | Предполагается ли повторное использование компонентов? | Нет | Нет | Да | Да | Да | Да |
| 10. | Являются ли достаточными ресурсы (время, деньги, инструменты, персонал)? | Нет | Нет | Нет | Нет | Да | Да |

Вычисления: 1 за каскадную, 2 за V-образную, 2 за RAD, 6 за инкрементную, 5 за быстрого прототипирования и 5 за эволюционную.

Итог: на основании результатов заполнения таблицы 2 подходящими являются каскадная V-образная и инкрементная модели.

В результате анализа различных моделей жизненного цикла разработки программного обеспечения, включая каскадную, V-образную, RAD, инкрементную, быструю прототипирование и эволюционную модели, были определены наиболее подходящие стратегии для нашего проекта. Каскадная и V-образная модели получили по 1 и 2 балла соответственно, инкрементная — 6, быстрая прототипирование и эволюционная — по 5 баллов.

Быстрое прототипирование позволяет создавать рабочие версии продукта для ранней обратной связи от пользователей, что помогает адаптировать его под их требования. Эволюционная модель обеспечивает поэтапное развитие и гибкость, позволяя вносить изменения на каждом этапе разработки.

Таким образом, наиболее подходящей моделью для нашего проекта является эволюционная модель.

**1.4 Инструменты разработки**

Для разработки проекта будет использована среда Visual Studio Code, обладающая функциями подсветки синтаксиса, автозавершения кода и интеграции с системами контроля версий, что упрощает процесс разработки. Она поддерживает множество расширений, позволяя адаптировать редактор под нужды проекта.

Основным языком программирования станет Python, известный своей простотой и выразительностью, что ускоряет разработку сложных систем. Для создания чат-бота будет использована библиотека Telebot, которая упрощает работу с API Telegram и позволяет сосредоточиться на логике приложения.

В качестве СУБД выбрана MySQL, обеспечивающая надежное хранение данных и масштабируемость. Библиотека pymysql для Python позволит эффективно взаимодействовать с базой данных, выполняя операции создания, чтения, обновления и удаления данных. MySQL также предлагает функции для обеспечения безопасности и производительности, что делает его отличным выбором для сложных приложений.

Иные инструменты, используемые при разработке и написании соответствующей документации:

* Web-ресурс DRAW.IO будет использоваться для разработки UML-диаграмм.
* Microsoft Office Word – для написания документации к программному продукту;

Разработка проекта будет происходить на компьютере со следующими параметрами:

* Процессор: AMD Ryzen™ 5 5500U (до 4,0 ГГц, 6 ядер, 12 потоков, 8 Мбайт кэш-памяти L3)
* Оперативная память: 16 Гбайт (2 x 8 Гбайт) ОЗУ, DDR4, 3200 МГц
* Видеокарта: AMD Radeon™ (встроенная)
* Операционная система: Windows 10 PRO

**1.5 Разработка плана работы над проектом**

Диаграмма Ганта — это эффективное визуальное представление графика работ, построенное в соответствии с планом проекта. Она позволяет наглядно отобразить задачи и последовательность их выполнения, что значительно облегчает управление проектом. На диаграмме представлены отрезки, размещенные вдоль временной оси, каждый из которых соответствует отдельной задаче или подзадаче. Такой формат позволяет не только увидеть сроки выполнения работ, но и оценить взаимосвязь между задачами, а также выявить возможные узкие места в процессе. Более детально ознакомиться с планом разработки проекта можно в Приложении Б.

**2. Проектирование задачи**

**2.1 Разработка структуры программного продукта, системы меню, навигации**

Структура чат-бота – это логическое построение всех функций и взаимодействий, обеспечивающее пользователю понимание доступных возможностей и навигации. Она включает приветственное сообщение, основное меню с категориями, подменю и интерактивные элементы, такие как кнопки и формы для ввода данных. Структура позволяет пользователю легко перемещаться между разделами, получать необходимую информацию и взаимодействовать с ботом. Структура представлена в Приложении В.

**2.2 Организация данных**

Модель данных – это абстрактная структура, которая упорядочивает элементы данных и определяет их взаимосвязи, а также связи с характеристиками объектов из реального мира. Она разрабатывается для того, чтобы продемонстрировать, какие типы данных используются и хранятся в системе, как они соотносятся друг с другом, а также как данные группируются и организуются, включая их форматы и атрибуты. Более детальную информацию о модели данных можно найти в Приложении Г.

**2.3 Разработка UML-диаграмм**

Использование диаграмм UML в проектировании систем позволяет создать более полное и четкое представление о динамическом поведении объектов и их взаимодействиях. Эти диаграммы служат важными инструментами для анализа и разработки, помогая командам разработчиков и аналитиков эффективно моделировать сложные системы и процессы. Они способствуют улучшению понимания требований к системе и обеспечивают более высокое качество конечного продукта. В проекте разработаны 3 диаграммы – диаграмма последовательности, диаграмма состояний, диаграмма классов.

Диаграмма последовательности — это один из типов диаграмм, используемых в языке моделирования UML. Она служит для визуализации взаимодействия между объектами в системе с точки зрения времени. Диаграмма последовательности иллюстрирует взаимодействие школьника с ботом, демонстрируя доступные функции и последовательность действий, которые он может выполнять. Диаграмма последовательности представлена в приложении Д.

Диаграмма деятельности – это один из типов диаграмм которая визуализирует последовательность действий, условий и решений, которые происходят в рамках определенной деятельности или процесса. С диаграммой деятельности можно ознакомится в приложении Е.

Диаграмма классов – это структурная диаграмма в UML, которая описывает классы, их атрибуты, методы и отношения между ними. Она предоставляет статическую модель системы, показывая, как различные классы взаимодействуют друг с другом и как они организованы. Диаграмма классов отображена в приложении Ж.

**2.4 Разработка пользовательского интерфейса**

Гайд по стилю Telegram-бота описывает визуальные и текстовые элементы интерфейса, обеспечивая единый и узнаваемый стиль, гармонирующий с дизайном Telegram. В качестве шрифтов используются Inter Medium, Semi Bold и Regular, где основной текст имеет размер 14–16 пикселей, а заголовки — 20–22 пикселя. Важные фрагменты выделяются жирным курсивом, а тональность текста варьируется: официальная информация оформляется нейтрально, а неформальная — ярко. Бот адаптируется под светлую и темную темы, сообщения имеют прозрачный фон, текст черный или темно-серый для светлой темы и белый для темной. Цвета кнопок подбираются с учетом контраста: темный текст на светлых кнопках и светлый на темных. Аватарка бота минималистична и круглой формы, создана нейросетью Midjourney, а изображения придерживаются стиля минимализма, плоских форм и мягких градиентов. Интерактивные элементы, такие как кнопки, создаются в стандартных размерах Telegram для удобства взаимодействия. Сообщения структурированы для легкости восприятия информации, с использованием абзацев и маркеров, а ключевые моменты выделяются заголовками или жирным шрифтом.

В приложении К можно ознакомиться с UX и UI макетами.

**3 Реализация**

**3.1 Руководство программиста**

Наиболее предпочтительным языком программирования для реализации проект стал Python, а также язык запросов SQL для выполнения операций с данными, в среде разработки Visual Studio Code.

Сначала были импортированы необходимые библиотеки для работы бота, такие как pymysql, telebot и модуль types.

Модуль types из библиотеки telebot содержит классы и функции для работы с типами данных Telegram API, включая сообщения, кнопки и инлайн-меню. Его импорт позволяет использовать эти элементы интерфейса в коде бота.

Для импортирования библиотек были использованы следующие строчки кода:

Import telebot

Import pymysql

From telebot import types

База данных играет ключевую роль в хранении информации о пользователях, анкетах и вопросах, обеспечивая надежное и структурированное управление данными. Она позволила эффективно организовывать, извлекать и обновлять информацию, что было критически важно для функционирования бота.

Далее приведен код для установления соединения с базой данных MySQL с использованием библиотеки pymysql:

connection = pymysql.connect(

host='127.0.0.1',

user='root',

password='1234',

database='psychobot'

)

В функции pymysql.connect() передаются параметры для подключения к базе данных. Параметр host указывает на сервер (127.0.0.1 — локальный компьютер), user задает имя пользователя (root), password содержит пароль (1234), а database указывает название базы данных (psychobot). После выполнения этой строки создается объект connection для выполнения SQL-запросов и взаимодействия с базой данных.

Для добавления пользователей в базу данных, в зависимости от их роли, была написана следующая функция:

def add\_user(first\_name, last\_name, telegram\_id, role):

try:

with connection.cursor() as cursor:

sql\_check = f»SELECT \* FROM {role} WHERE TelegramID = %s»

cursor.execute(sql\_check, (telegram\_id,))

result = cursor.fetchone()

if result:

return «exists», result[1], result[2]

else:

sql = f»INSERT INTO {role} (FirstName, LastName, TelegramID) VALUES (%s, %s, %s)»

cursor.execute(sql, (first\_name, last\_name, telegram\_id))

connection.commit()

return «added», first\_name, last\_name

except Exception as ex:

print(«Ошибка при добавлении пользователя!»)

print(ex)

return «error», None, None

Функция add\_user добавляет пользователя в базу данных в зависимости от его роли (студент, психолог или администратор). Она принимает имя, фамилию, Telegram ID и роль. Сначала проверяется, существует ли пользователь с данным Telegram ID. Если найден, возвращается статус «exists» с именем и фамилией. Если нет, выполняется SQL-запрос для добавления нового пользователя, и при успешном добавлении возвращается статус «added». В случае ошибки выводится сообщение и возвращается статус «error».

Теперь время рассказать про обработчик start\_bot, который приветствует пользователей и определяет их роли в системе при получении команды /start.

@bot.message\_handler(commands=[«start»])

@check\_blocked

def start\_bot(message):

telegram\_id = str(message.chat.id)

role = None

user\_info = None

Сначала мы получаем telegram\_id из сообщения пользователя. Затем переменные role и user\_info остаются пустыми, так как мы ещё не знаем, зарегистрирован ли пользователь.

Далее открываем соединение с базой данных и поочерёдно проверяем три таблицы (Students, Psychologists, Admins):

try:

with connection.cursor() as cursor:

for table in [«Students», «Psychologists», «Admins»]:

sql\_check = f»SELECT FirstName, LastName FROM {table} WHERE TelegramID = %s»

cursor.execute(sql\_check, (telegram\_id,))

result = cursor.fetchone()

if result:

role = table

user\_info = result

break

Если запись найдена, то мы сохраняем её в переменные role и user\_info, после чего прекращаем проверку.

Если возникает ошибка, она обрабатывается в блоке except:

except Exception as ex:

print(«Ошибка при проверке пользователя!»)

print(ex)

bot.send\_message(message.chat.id, «Произошла ошибка при обработке. Попробуйте позже.»)

return

В этом случае пользователь получит сообщение об ошибке, а сама ошибка будет выведена в консоль.

Если пользователь найден, отправляем ему приветственное сообщение с указанием его роли:

if role:

first\_name, last\_name = user\_info

bot.send\_message(

message.chat.id,

f»Добро пожаловать обратно, {first\_name} {last\_name}! Вы зарегистрированы под ролью {role}.»,

reply\_markup=get\_role\_keyboard(role)

)

Здесь мы приветствуем пользователя по имени и предоставляем клавиатуру с функциями, соответствующими его роли.

Если пользователь не найден, ему предлагается выбрать свою роль:

else:

keyboard\_roles = types.ReplyKeyboardMarkup(resize\_keyboard=True)

keyboard\_roles.add(

types.KeyboardButton(«Я ученик»),

types.KeyboardButton(«Я психолог»),

types.KeyboardButton(«Я администратор»)

)

bot.send\_message(

message.chat.id,

«Добро пожаловать в Кабинет Школьного Психолога!\nДля продолжения выберите свою роль:»,

reply\_markup=keyboard\_roles

)

Здесь мы предоставляем три варианта ролей, из которых пользователь может выбрать подходящую.

В обработчиках кнопок есть декоратор check\_blocked:

def check\_blocked(func):

def wrapper(message):

user\_id = message.from\_user.id

if is\_user\_blocked(user\_id):

bot.send\_message(user\_id, «Ваш аккаунт был заблокирован! Вы не можете пользоваться ботом.»)

return

return func(message)

return wrapper

Декоратор check\_blocked проверяет, заблокирован ли пользователь, перед выполнением основной функции обработчика. Он получает ID пользователя, использует функцию is\_user\_blocked для проверки блокировки и, если пользователь заблокирован, отправляет сообщение о запрете доступа. Если не заблокирован, декоратор вызывает основную функцию.

Рассмотрим обработчик для кнопки «Получить анонимную помощь”.

Обработчик срабатывает, когда ученик выбирает опцию «Получить анонимную помощь».

@bot.message\_handler(func=lambda message: message.text == «Получить анонимную помощь») # Проверяем текст сообщения.

@check\_blocked

def anonymous\_help(message):

user\_id = message.chat.id

Здесь мы определяем функцию, которая запускается, если пользователь нажимает на соответствующую кнопку.

Пользователь получает сообщение с инструкцией о том, как отправить анонимное обращение:

bot.send\_message(

user\_id,

«Похоже, что вы нуждаетесь в анонимной помощи.\n»

«Пришлите анонимное сообщение, и психолог ответит на него.»

)

bot.register\_next\_step\_handler(message, handle\_anon\_response) # Ждём следующего сообщения от пользователя.

Сообщение объясняет пользователю, что он может отправить текст, который будет передан психологу. После этого бот переключается на ожидание следующего шага — обработку текста обращения.

Следующая функция обрабатывает текст анонимного сообщения:

def handle\_anon\_response(message):

user\_id = message.chat.id

anon\_message\_text = message.text

Здесь мы фиксируем ID пользователя и текст его сообщения. Эти данные будут сохраняться в базе данных.

Попробуем сохранить сообщение в базе данных. Если сохранение успешно, пользователь получает подтверждение:

try:

with connection.cursor() as cursor:

sqlInsertAnonMessage = «««

INSERT INTO AnonymousMessages (StudentID, MessageText)

VALUES ((SELECT StudentID FROM Students WHERE TelegramID = %s), %s)

««« # SQL-запрос для добавления анонимного сообщения.

cursor.execute(sqlInsertAnonMessage, (user\_id, anon\_message\_text)) # Выполняем запрос с параметрами.

connection.commit()

bot.send\_message(

user\_id,

«Ваше сообщение отправлено, психолог скоро на него ответит.»

)

В этом блоке SQL-запрос сначала находит StudentID по Telegram ID пользователя, а затем сохраняет сообщение в таблице AnonymousMessages.

Если возникает ошибка, она обрабатывается в блоке except. В этом случае отправляем пользователю сообщение об ошибке:

except Exception as ex:

bot.send\_message(

user\_id,

f»Произошла ошибка: {ex}\nПопробуйте повторить позже!»

)

Здесь ошибка логируется и пользователь узнаёт, что произошла проблема. Ему предлагается повторить попытку позже.

**3.2 Организация данных**

В этом разделе описывается структура и назначение данных, используемых в проекте. Все данные можно условно разделить на три категории: входные, выходные и постоянные. Входные данные представляют собой информацию, поступающую в систему для обработки, включая пользовательские вводы и данные из внешних источников. Выходные данные — это результаты обработки, которые система предоставляет пользователям или другим системам в виде отчетов, графиков и уведомлений. Постоянные данные хранятся в системе на длительный срок и обеспечивают целостность и непрерывность работы проекта.

**3.2.1 Типы данных и их назначение**

**3.2.1.1 Константы**

TOKEN — токен авторизации Telegram-бота. Используется для взаимодействия с API Telegram.

**3.2.1.2 Переменные**

TOKEN — токен авторизации Telegram-бота. Используется для взаимодействия с API Telegram.

bot — объект, представляющий Telegram-бота. Управляет отправкой и получением сообщений через API.

connection — объект подключения к базе данных MySQL, который используется для выполнения SQL-запросов.

user\_states – словарь, в котором ключом является ID пользователя, а значением — состояние пользователя. Это состояние может включать в себя выбранную роль, текущий этап анкеты или теста, что позволяет эффективно отслеживать прогресс и взаимодействие каждого пользователя с системой.

questionnaire\_title — название анкеты, которую создает пользователь.

question\_count – количество вопросов в анкете.

**3.2.1.3 База данных**

Для хранения постоянной информации используется система управления базами данных MySQL, состоящая из нескольких таблиц.

Таблица Students содержит поля StudentID (уникальный идентификатор учащегося), FirstName, LastName и TelegramID (уникальный идентификатор пользователя в Telegram) для хранения данных об учащихся.

Таблица Psychologists имеет аналогичную структуру и хранит информацию о психологах с теми же полями.

Таблица Admins также повторяет структуру таблицы Students, но предназначена для хранения информации об администраторах.

Таблица Content хранит публикации психологов и включает поля ContentID (уникальный идентификатор контента) и ContentData (текст публикаций) для доступа учащихся к материалам.

Таблица Block содержит информацию о заблокированных пользователях с полями blockID (уникальный идентификатор блокировки) и blockuser\_id (идентификатор заблокированного пользователя).

Таблица AnonymousMessages предназначена для хранения анонимных обращений и ответов, включая поля MessageID (уникальный идентификатор сообщения), StudentID, PsychologistID, MessageText, ResponseText и CreatedAt (дата и время отправки).

Таблица questionnaires хранит анкеты с полями id (уникальный идентификатор анкеты), title (название анкеты) и published (флаг публикации).

Таблица questions содержит вопросы анкет с полями id (уникальный идентификатор вопроса), questionnaire\_id (идентификатор анкеты) и question (текст вопроса).

Таблица answers предназначена для хранения ответов учащихся на вопросы анкет, включая поля id (уникальный идентификатор ответа), questionnaire\_id, student\_id и answer (текст ответа).

**3.2.1.4 Входные данные**

Входные данные, обрабатываемые системой, включают информацию, вводимую пользователями, а также данные, получаемые из базы данных.

Пользователи взаимодействуют с ботом через команды Telegram, такие как /start, выбор ролей, а также предоставляют ответы на тесты и анкеты. Кроме того, они могут отправлять текстовые сообщения и анонимные обращения, что позволяет им делиться своими мыслями и вопросами в удобной форме. Психологи, в свою очередь, создают содержание статей и анкет, которые также становятся частью входных данных.

Система получает данные из базы, включая списки анкет, вопросов и контента, что обеспечивает актуальность информации. Важным аспектом является состояние пользователей, включая информацию о блокировке, что позволяет эффективно управлять доступом и безопасностью. Это обеспечивает качественное взаимодействие между пользователями и ботом.

**3.2.1.5 Выходные данные**

Выходные данные системы включают сообщения, отправляемые пользователям, и изменения в базе данных. Система отправляет приветственные и информационные уведомления, помогающие пользователям ориентироваться в функционале бота, а также результаты тестов, таких как уровень тревожности, что способствует пониманию их состояния. Публикации, включая статьи, предоставляют доступ к полезной информации, а ответы на анонимные обращения обеспечивают обратную связь и поддержку.

Кроме сообщений, система вносит изменения в базу данных, регистрируя пользователей в таблицах Students, Psychologists и Admins для отслеживания их активности. Анкеты и вопросы сохраняются в таблицах questionnaires и questions, а записи ответов пользователей фиксируются в таблице answers, что позволяет анализировать данные и улучшать качество услуг

**3.2.1.6 Постоянные данные**

Постоянные данные в базе данных содержат основную информацию, необходимую для функционирования системы. В первую очередь, это данные о пользователях — учащихся, психологах и администраторах, что позволяет системе управлять ролями и обеспечивать соответствующий доступ к функционалу.

Также хранятся публикации, анкеты, вопросы и ответы, которые являются ключевыми для взаимодействия пользователей с системой, предоставляя ресурсы для обучения и самопознания. Важным аспектом являются данные о состоянии заблокированных пользователей, что позволяет контролировать доступ и обеспечивать безопасность, предотвращая нежелательные взаимодействия и поддерживая порядок в сообществе.

**3.2.2 Используемые структуры данных**

**3.2.2.1 Словари**

Tests — используется для хранения вопросов тестов.

user\_states — хранит состояние взаимодействия с пользователем.

**3.2.2.2 Списки**

current\_questionnaire — временное хранилище вопросов новой анкеты.

answers – список который хранит варианты ответа для прохождения теста.

**3.3 Назначение данных в проекте**

Данные организованы для удобного взаимодействия пользователей с ботом, включая регистрацию, выбор ролей, тесты и анкеты. Все результаты сохраняются в базе данных для эффективного управления информацией. Реализовано управление доступом, включая блокировку пользователей через таблицу Block, что обеспечивает безопасность. Такой подход создает интуитивно понятный интерфейс и надежное хранение данных, улучшая обслуживание пользователей.

**4 Тестирование**

**4.1 Тесты на использование**

В процессе разработки чат-бота для Telegram было выявлено множество ошибок и недоработок, которые были успешно устранены на этапе реализации проекта. После завершения этапа испытаний программы было проведено детальное функциональное тестирование.

Разработанные тест-кейсы и статус их выполнения представлены в приложение К.

Расписание работ над проектом представлено в таблице 6.

Таблица 6 – Расписание работ над проектом

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Дата** | **Деятельность** | **Продолжительность, ч** |
| Шаматульский Никита | 02.12.2024 | Разработка тестов | 1 |
| Шаматульский Никита | 03.12.2024 | Тестирование чат-бота | 3 |
| Шаматульский Никита | 04.12.2024 | Составление отчетов о найденных дефектах | 2 |
| Шаматульский Никита | 04.12.2024 | Исправление найденных ошибок | 2 |
| Шаматульский Никита | 06.12.2024 | Проведение регрессионного тестирования | 2 |
| Шаматульский Никита | 06.12.2024 | Составление отчета о результатах тестирования | 3 |

**4.2 Отчёт о результатах тестирования**

Элементы программы были проверены, и было установлено, что все они работают правильно и выполняют задачи, указанные в процедурах.

Статистика по всем дефектам представлена в таблице 7.

Пройденные тесты можно найти в приложении Л.

Таблица 7 – Статистика по всем дефектам

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Статус | Количество | Важность | | | |
| Низкая | Средняя | Высокая | Критическая |
| Найдено | 0 | 0 | 0 | 0 | 0 |
| Исправлено | 0 | 0 | 0 | 0 | 0 |
| Проверено | 0 | 0 | 0 | 0 | 0 |
| Открыто заново | 0 | 0 | 0 | 0 | 0 |
| Отклонено | 0 | 0 | 0 | 0 | 0 |

**5 Руководство пользователя**

Цель проекта — создать удобный инструмент для взаимодействия учащихся, школьных психологов, улучшая психологическую поддержку и образовательные материалы. Внедряются функции анонимных консультаций, анкет и персонализированного доступа к ресурсам. Телеграм-бот ориентирован на широкую аудиторию: учащихся, педагогов-психологов и администраторов. Он протестирован на различных устройствах и версиях Telegram, что подтверждает его стабильность и адаптивность. Простота интерфейса и продуманная структура обеспечивают легкость использования для школьников и взрослых.

**5.1 Руководство системного программиста**

Для запуска телеграм-бота необходимо запустить мессенджер Telegram.

**5.2 Запуск программы**

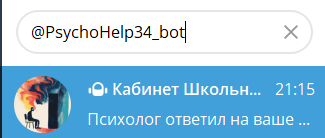
Чтобы запустить чат-бот, необходимо в поисковой строке ввести тег бота (@PsychoHelp34\_bot) как представлено на рисунке 1, перейти в диалог с ботом, и нажать кнопку старт, в соответствии с рисунком 2.

Рисунок 1 – Ввод названия бота в поисковую строку Telegram

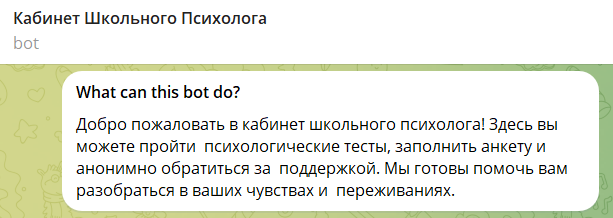
****

Рисунок 2 – Стартовое меню чат-бота

**5.3 Инструкция по работе с программой**

После нажатия на кнопку «Старт» начнется процесс авторизации, в ходе которого бот запросит у пользователя его роль, а затем фамилию и имя. В зависимости от роли пользователю будут предложены несколько кнопок, Расмотрим кнопки для роли учащегося, которые отображены на рисунке 3.

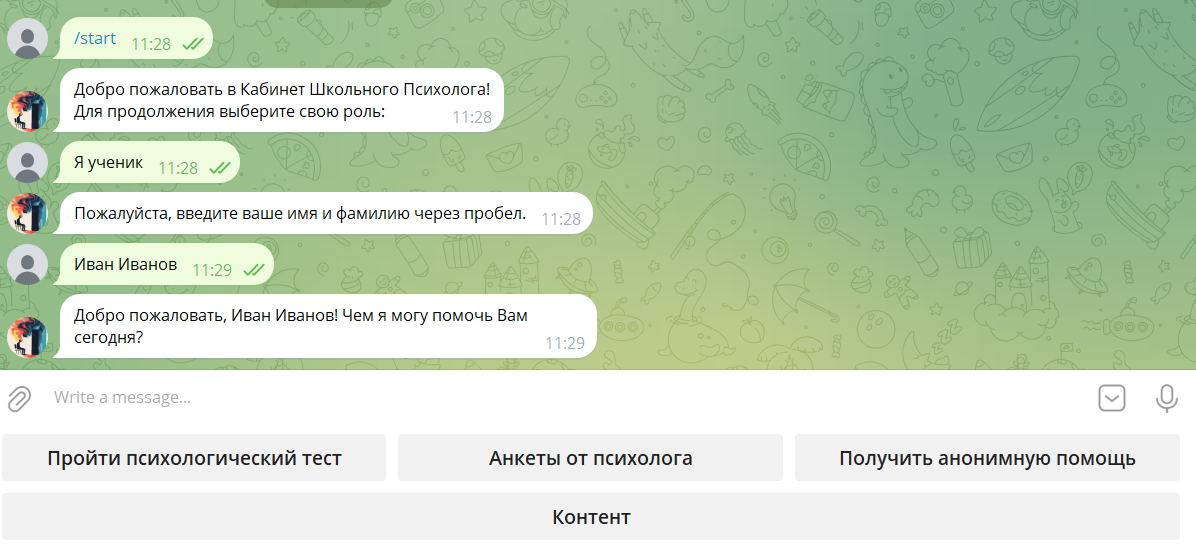


Рисунок 3 – Кнопки учащегося

Для рассмотрим кнопку «Пройти психологический тест». Эта кнопка позволит учащемуся пройти один из психологических тестов на выбор. Выбрать психологический тест ученик может на рисунке 4.

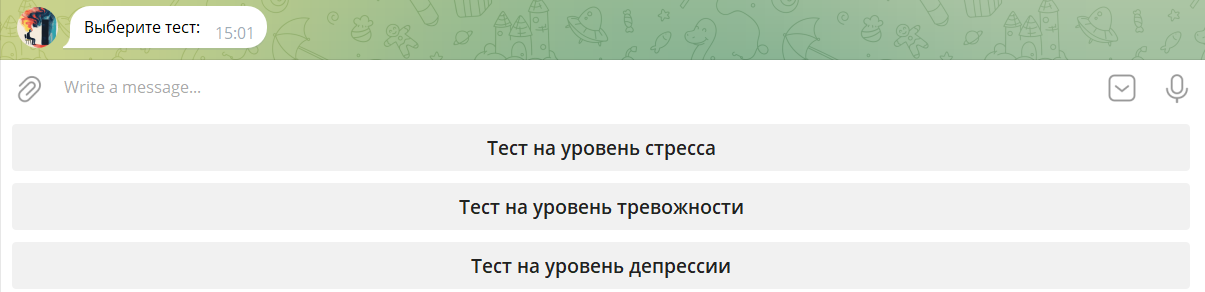


Рисунок 4 – Выбор психологического теста

Тест проводится в виде диалога с ботом, где пользователь отвечает на вопросы. В конце теста бот отправит оценку теста. Процесс прохождения теста представлен на рисунке 5.

Рисунок 5 – Прохождение тестов от бота

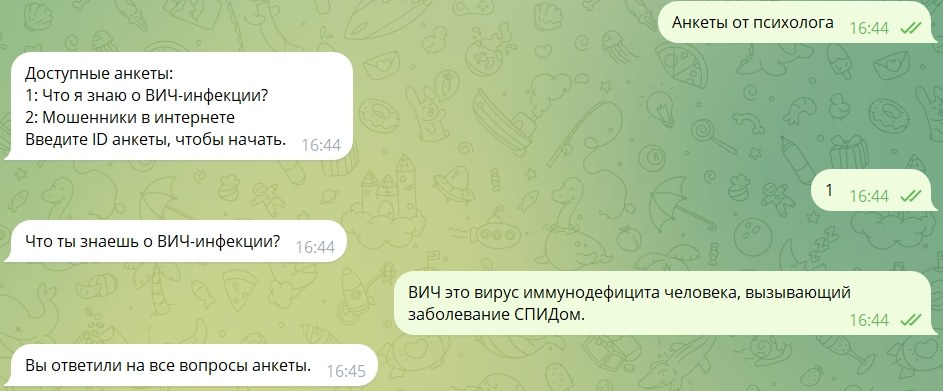
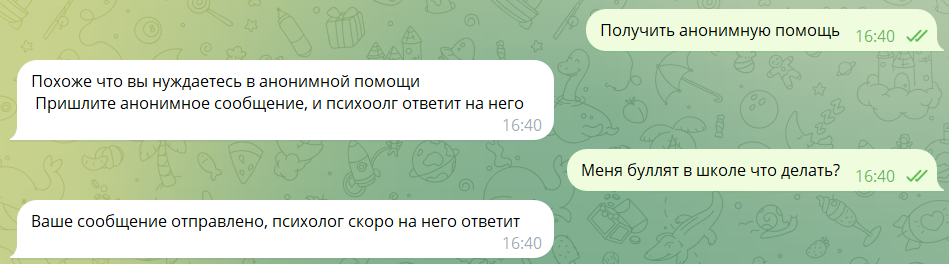
Далее рассмотрим кнопку «Анкеты от психолога». В ней ученик может пройти анкеты от психолога, представленную на рисунке 6.

Рисунок 6 – Прохождение анкеты от психолога

Кнопка «Получить анонимную помощь» позволяет учащимся обратиться к психологу без раскрытия личности, создавая безопасное пространство для обсуждения личных проблем. Учащиеся могут воспользоваться этой функцией в любое время, что делает поддержку более доступной и комфортной. Рисунок 7 иллюстрирует простой процесс анонимного обращения.

Рисунок 7 – Обращение за анонимной помощью

Последняя кнопка у учащегося – это контент, где он может просматривать разные статьи и новости. Посмотреть реализацию кнопки можно на рисунке 8.

Рисунок 8 – Кнопка «Контент»

**Заключение**

В заключение практики, посвященной разработке и сопровождению программного обеспечения, можно отметить успешную реализацию чат-бота, который стал важным элементом проекта. Бот был разработан с учетом удобства пользователей и интуитивно понятного интерфейса, что позволило обеспечить высокий уровень взаимодействия с конечными пользователями.

В ходе работы над проектом были учтены все возможные исключительные ситуации, что позволило минимизировать количество ошибок и сбоев в работе бота. Благодаря тщательному тестированию, функциональность электронного ресурса была подтверждена, и он успешно выполняет поставленные задачи.

Тем не менее, существует потенциал для дальнейшего развития данного программного продукта. Возможные направления для улучшения включают добавление новых функций, таких как интеграция с внешними сервисами, расширение возможностей взаимодействия с пользователями через дополнительные команды и настройки, а также улучшение аналитики и отчетности по использованию бота.

В целом, реализация чат-бота в рамках практики позволила не только приобрести новые знания и навыки в области разработки программного обеспечения, но и создать законченный и функциональный продукт, который может быть полезен пользователям. Опыт, полученный в ходе работы над проектом, будет полезен для дальнейшей карьеры в сфере разработки программного обеспечения.

**Список использованных источников**

1. Stack Overflow [электронный ресурс]. – Электронные данные. – Режим доступа: <https://stackoverflow.com/> – Дата доступа: 15.11.2024;
2. Хабр Q&A [электронный ресурс]. – Электронные данные. – Режим доступа: <https://qna.habr.com/> – Дата доступа: 17.11.2024;
3. MySQL documentation [электронный ресурс]. – Электронные данные. – Режим доступа: <https://dev.mysql.com/doc/> – Дата доступа: 03.12.2024;
4. Telebot documentation [электронный ресурс]. – Электронные данные. – Режим доступа: <https://pytba.readthedocs.io/en/latest/> – Дата доступа: 03.12.2024;  
   ПОПРОСИТЬ У ПЗТ-40 ТАМ СОВСЕМ ДРУГОЕ

**Приложение А**   
Диаграмма вариантов использования

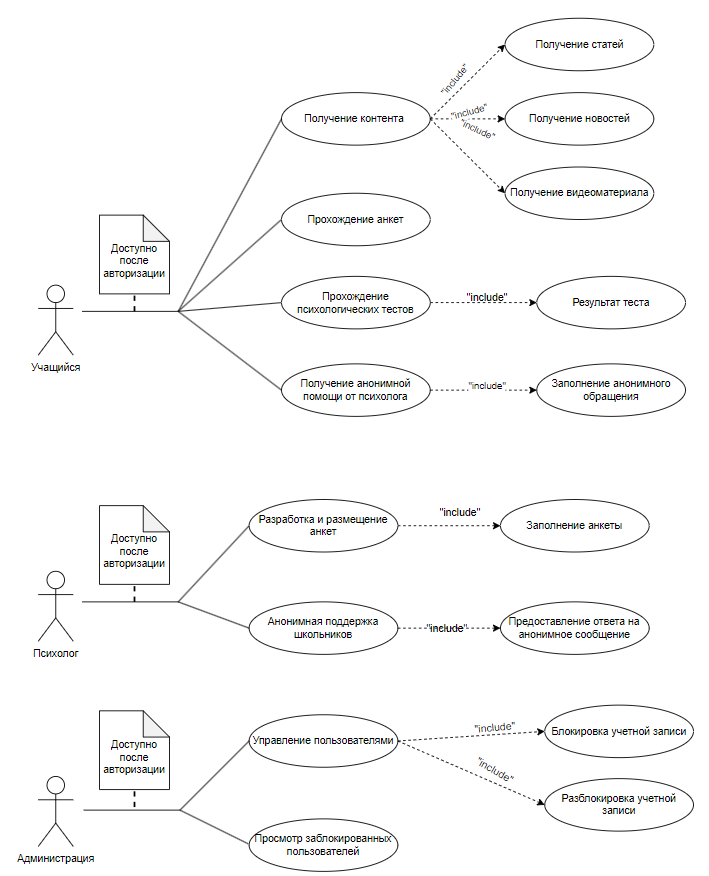


Рисунок А.1 – Диаграмма вариантов использования

**Приложение Б**  
Диаграмма Ганта

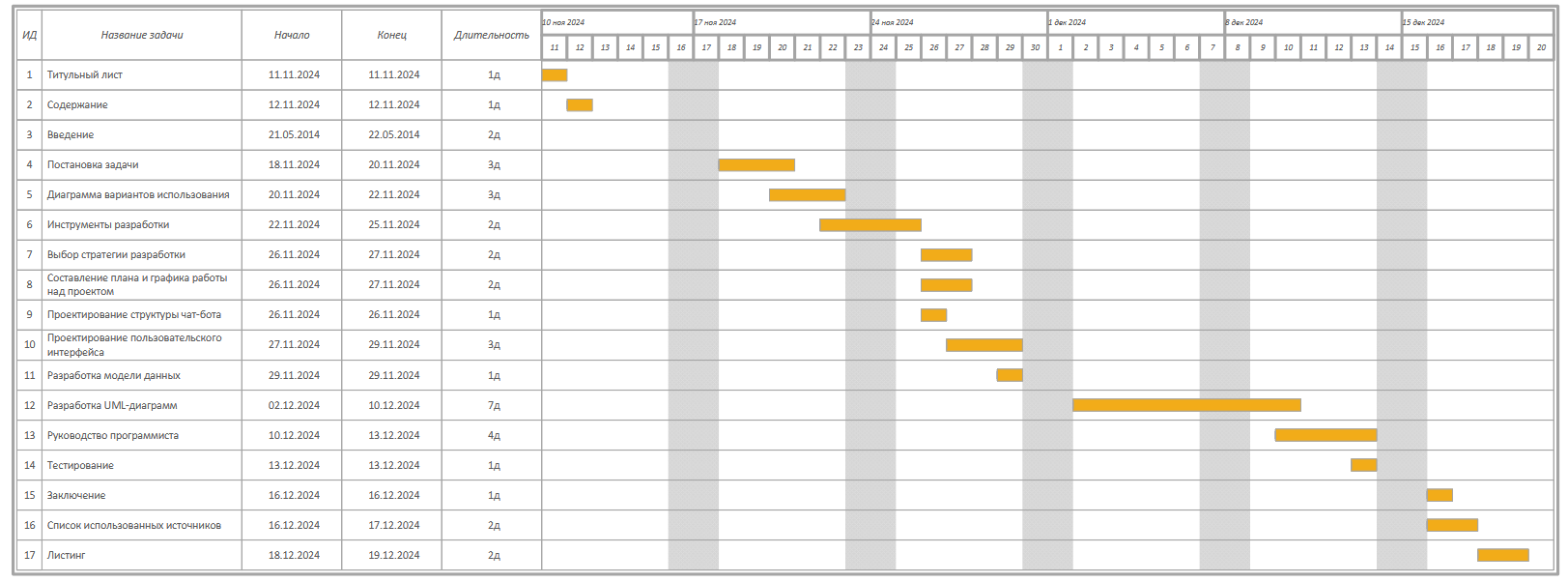


Рисунок Б.1 – Диаграмма Ганта

**Приложение В**   
Структура чат-бота

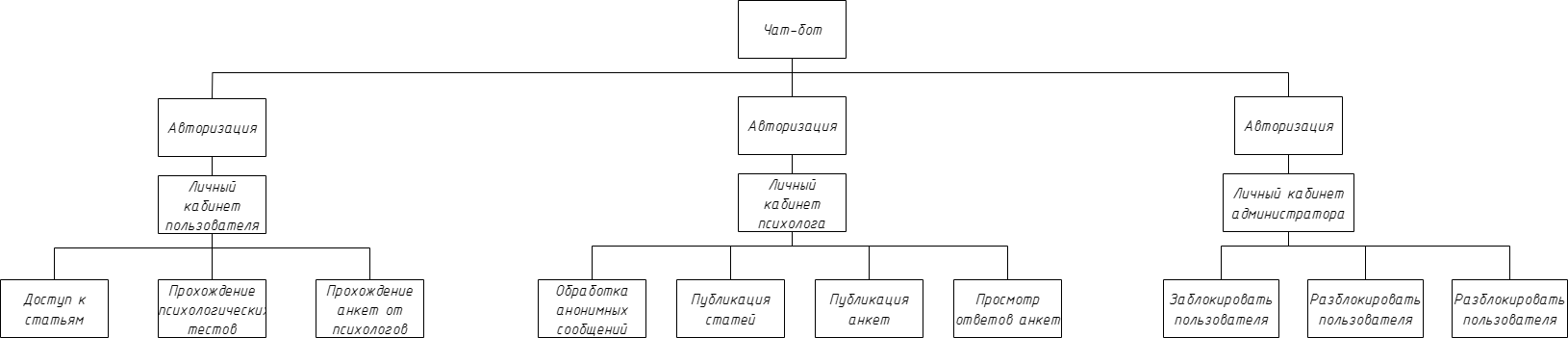


Рисунок В.1 – Структура чат-бота

**Приложение Г**  
Модель данных

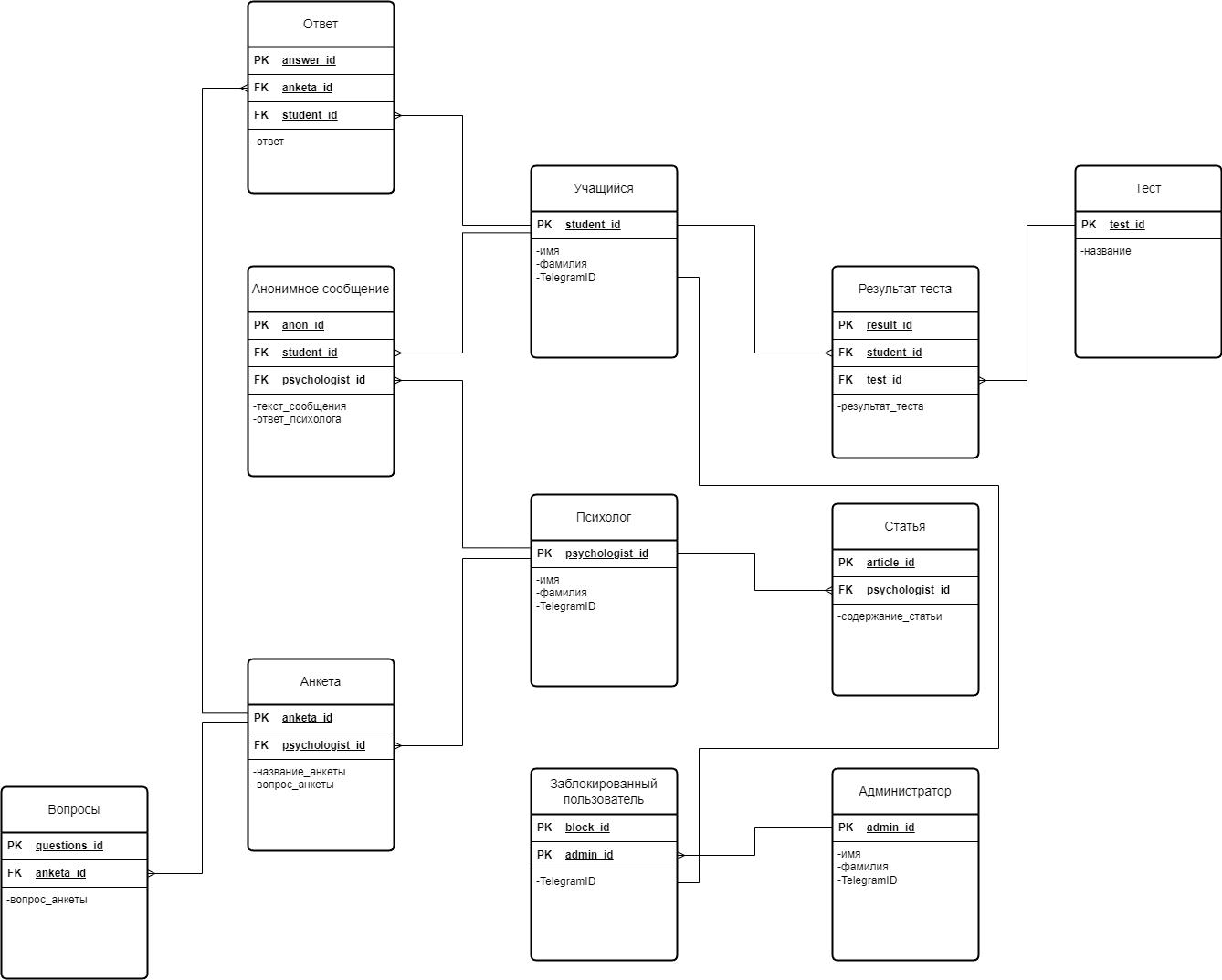


Рисунок Г.1 – Модель данных

**Приложение Д**   
Диаграмма последовательности

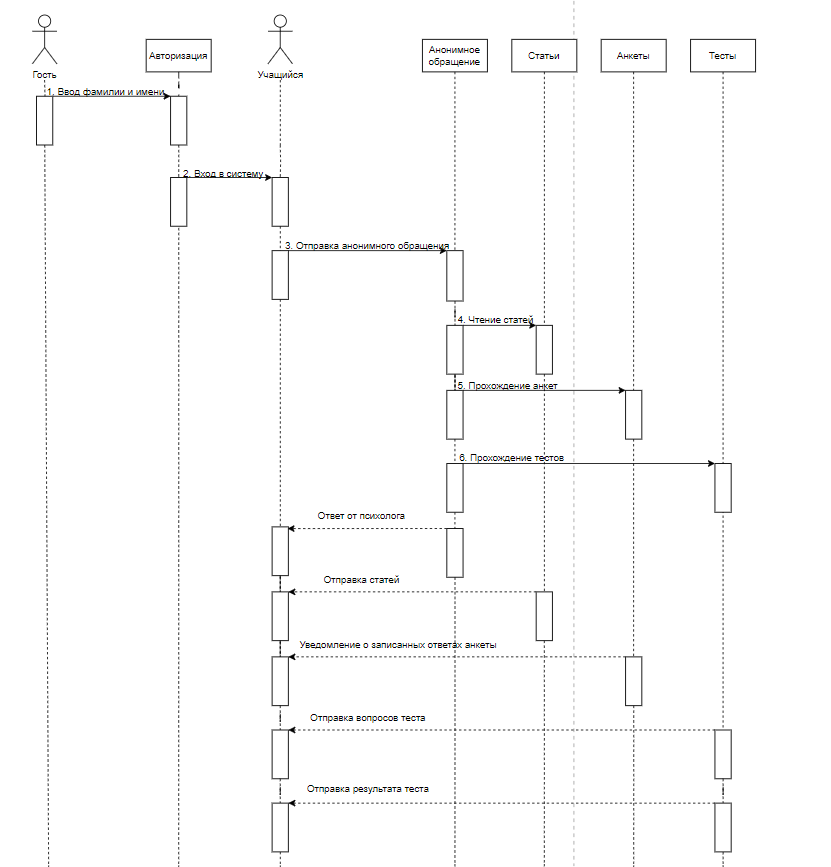


Рисунок Д.1 – Диаграмма последовательности

**Приложение Е**   
Диаграмма деятельности

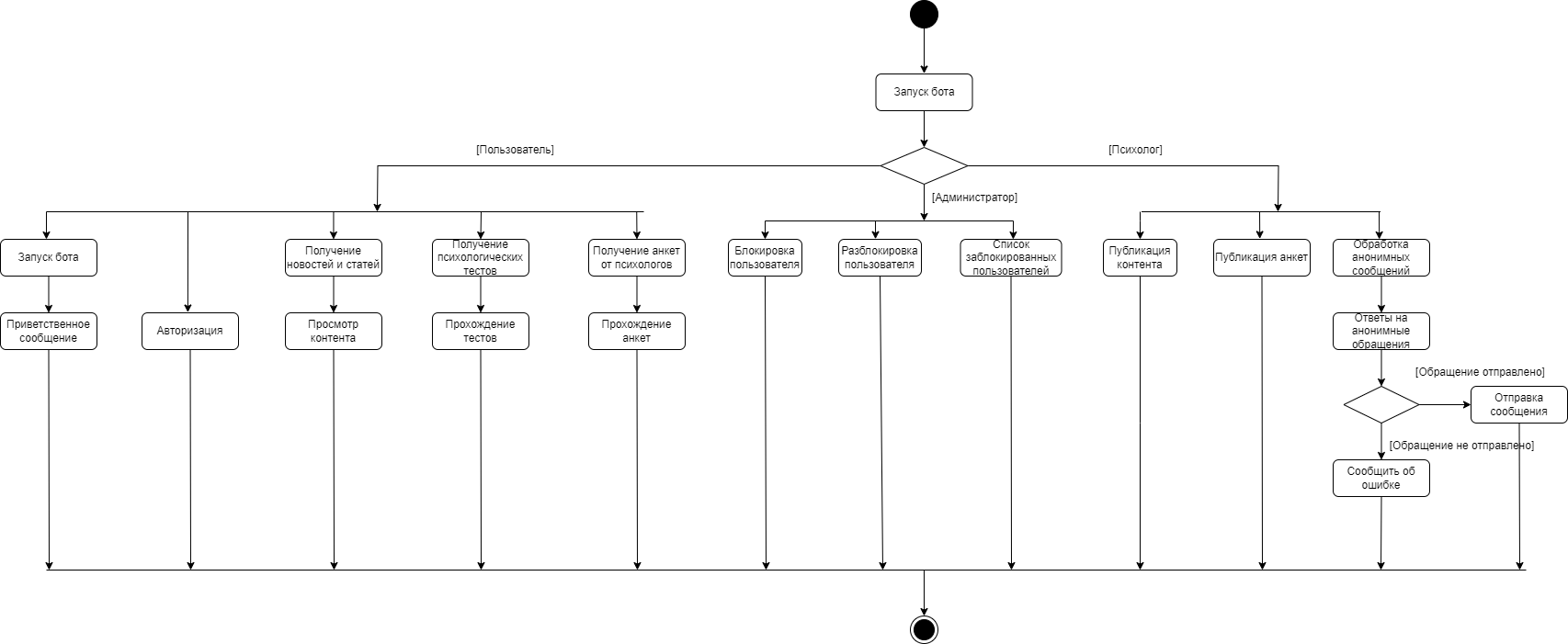


Рисунок Е.1 – Диаграмма деятельности

**Приложение Ж**   
Диаграмма классов

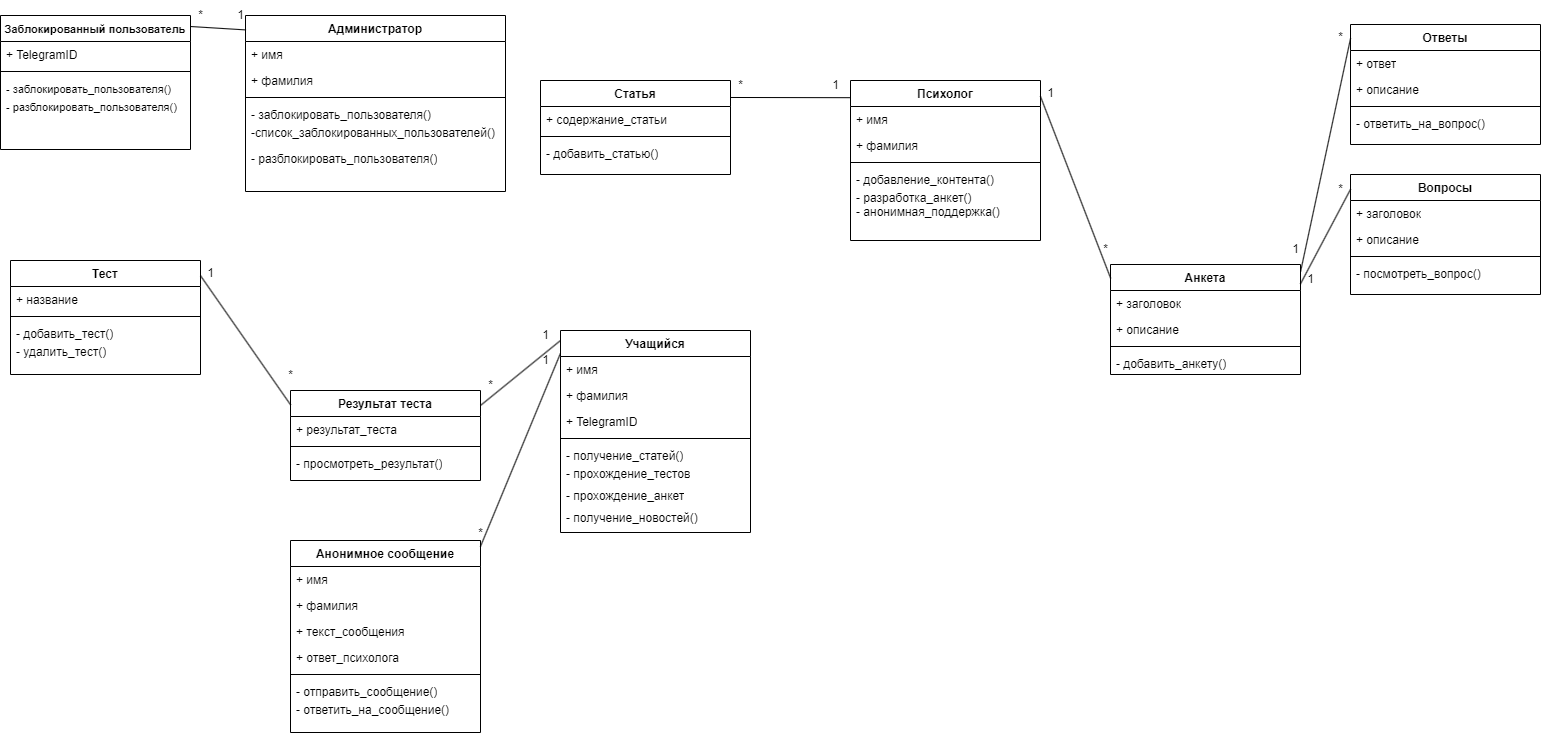


Рисунок Ж.1 – Диаграмма классов

**Приложение К**  
Пользовательский интерфейс программы

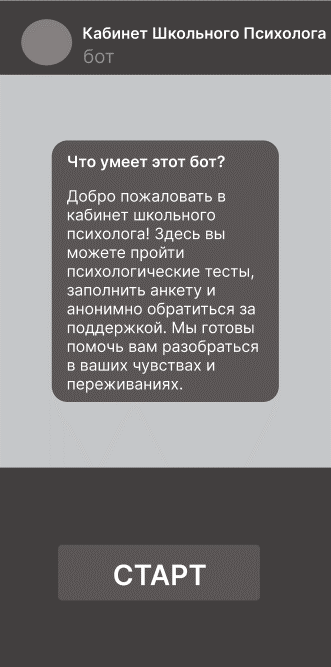
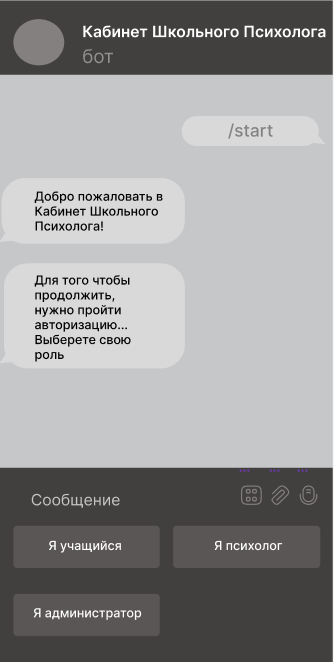
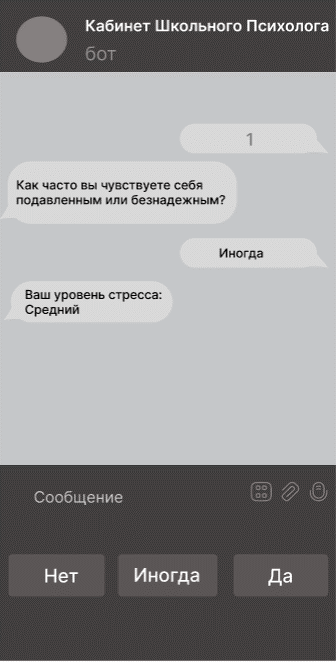
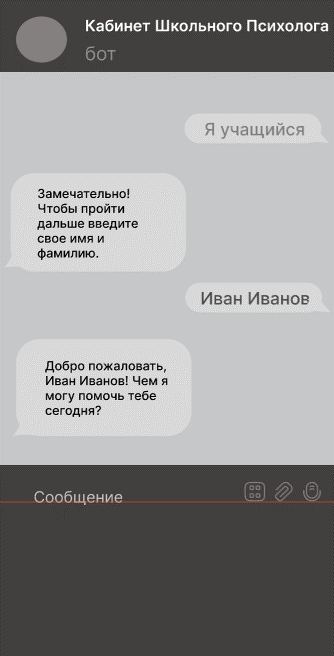


Рисунок К.2 – UX прототипы бота

Рисунок К.1 - UX прототипы первых стартовых страниц

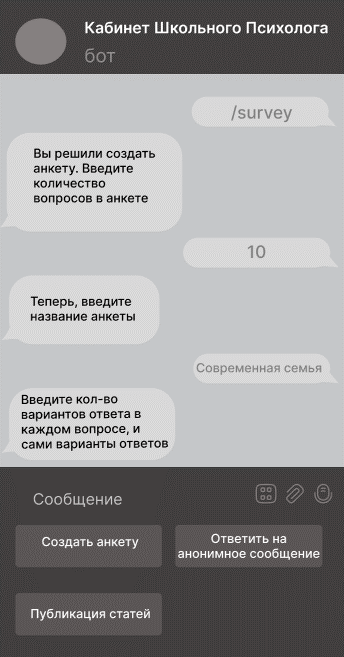
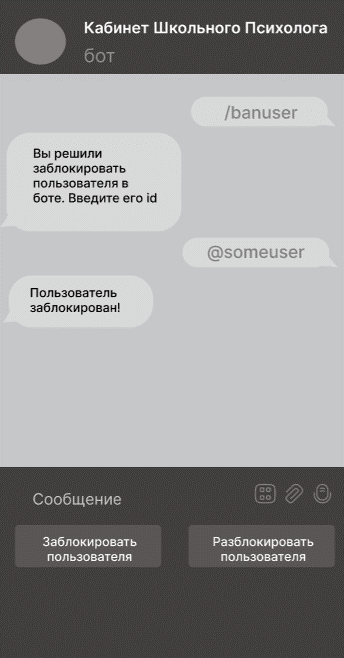


Рисунок К.3 – UX прототипы страниц психолога и администратора

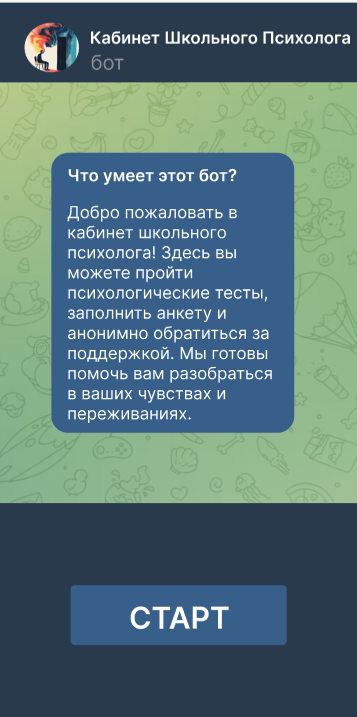
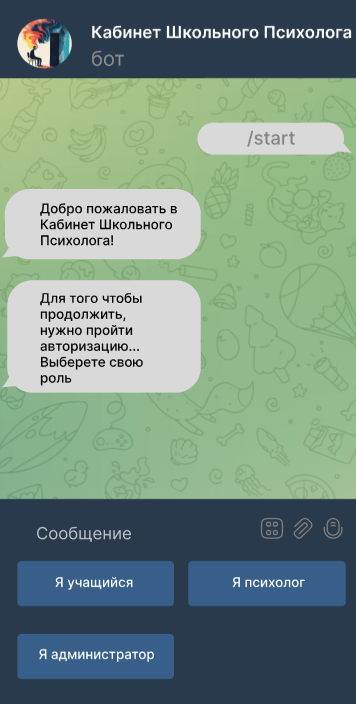


Рисунок К.4 – UI прототипы

**Приложение Л**   
Тест-кейсы

**Приложение П**  
Листинг приложения

import telebot

import pymysql

from telebot import types

TOKEN = "YOUR\_TOKEN\_HERE"

bot = telebot.TeleBot(TOKEN)

print("Бот запущен")

user\_states = {}

connection = pymysql.connect(

host='127.0.0.1',

user='root',

password='1234',

database='psychobot'

)

# Инициализация таблиц

def initialize\_database():

queries = [

'''

CREATE TABLE IF NOT EXISTS Students(

StudentID INT AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(100),

LastName VARCHAR(100),

TelegramID VARCHAR(100) UNIQUE

)

''',

'''

CREATE TABLE IF NOT EXISTS Psychologists(

PsychologistID INT AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(100),

LastName VARCHAR(100),

TelegramID VARCHAR(100) UNIQUE

)

''',

'''

CREATE TABLE IF NOT EXISTS Admins(

AdminID INT AUTO\_INCREMENT PRIMARY KEY,

FirstName VARCHAR(100),

LastName VARCHAR(100),

TelegramID VARCHAR(100) UNIQUE

)

''',

'''

CREATE TABLE IF NOT EXISTS Content(

ContentID INT AUTO\_INCREMENT PRIMARY KEY,

ContentData TEXT

)

''',

'''

CREATE TABLE IF NOT EXISTS Block(

blockID INT AUTO\_INCREMENT PRIMARY KEY,

blockuser\_id INT UNIQUE

)

''',

'''

CREATE TABLE IF NOT EXISTS AnonymousMessages (

MessageID INT AUTO\_INCREMENT PRIMARY KEY,

StudentID INT,

PsychologistID INT DEFAULT NULL,

MessageText TEXT,

ResponseText TEXT DEFAULT NULL,

CreatedAt TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT fk\_student FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

CONSTRAINT fk\_psychologist FOREIGN KEY (PsychologistID) REFERENCES Psychologists(PsychologistID)

)

''',

'''

CREATE TABLE IF NOT EXISTS questionnaires (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255),

question TEXT NULL,

published BOOLEAN DEFAULT FALSE

)

''',

'''

CREATE TABLE IF NOT EXISTS answers (

id INT AUTO\_INCREMENT PRIMARY KEY,

questionnaire\_id INT,

student\_id INT,

answer TEXT,

FOREIGN KEY (questionnaire\_id) REFERENCES questionnaires(id)

)

''',

'''

CREATE TABLE IF NOT EXISTS questions (

id INT AUTO\_INCREMENT PRIMARY KEY,

questionnaire\_id INT,

question TEXT,

FOREIGN KEY (questionnaire\_id) REFERENCES questionnaires(id)

)

'''

]

try:

with connection.cursor() as cursor:

for query in queries:

cursor.execute(query)

connection.commit()

print("Успешное подключение к БД! Все таблицы инициализированы..")

except Exception as ex:

print("Ошибка при подключении к БД!", ex)

initialize\_database()

# Проверка блокировки пользователя

def check\_blocked(func):

def wrapper(message):

user\_id = message.from\_user.id

if is\_user\_blocked(user\_id):

bot.send\_message(user\_id, "Ваш аккаунт был заблокирован! Вы не можете пользоваться ботом.")

return

return func(message)

return wrapper

# Функция добавления пользователя

def add\_user(first\_name, last\_name, telegram\_id, role):

try:

with connection.cursor() as cursor:

sql\_check = f"SELECT \* FROM {role} WHERE TelegramID = %s"

cursor.execute(sql\_check, (telegram\_id,))

result = cursor.fetchone()

if result:

return "exists", result[1], result[2]

else:

sql = f"INSERT INTO {role} (FirstName, LastName, TelegramID) VALUES (%s, %s, %s)"

cursor.execute(sql, (first\_name, last\_name, telegram\_id))

connection.commit()

return "added", first\_name, last\_name

except Exception as ex:

print("Ошибка при добавлении пользователя!", ex)

return "error", None, None

@bot.message\_handler(commands=["start"])

@check\_blocked

def start\_bot(message):

telegram\_id = str(message.chat.id)

role, user\_info = None, None

try:

with connection.cursor() as cursor:

for table in ["Students", "Psychologists", "Admins"]:

sql\_check = f"SELECT FirstName, LastName FROM {table} WHERE TelegramID = %s"

cursor.execute(sql\_check, (telegram\_id,))

result = cursor.fetchone()

if result:

role, user\_info = table, result

break

except Exception as ex:

print("Ошибка при проверке пользователя!", ex)

bot.send\_message(message.chat.id, "Произошла ошибка при обработке. Попробуйте позже.")

return

if role:

first\_name, last\_name = user\_info

bot.send\_message(

message.chat.id,

f"Добро пожаловать обратно, {first\_name} {last\_name}! Вы зарегистрированы под ролью {role}.",

reply\_markup=get\_role\_keyboard(role)

)

else:

present\_role\_options(message)

def present\_role\_options(message):

keyboard\_roles = types.ReplyKeyboardMarkup(resize\_keyboard=True)

keyboard\_roles.add(

types.KeyboardButton("Я ученик"),

types.KeyboardButton("Я психолог"),

types.KeyboardButton("Я администратор")

)

bot.send\_message(

message.chat.id,

"Добро пожаловать в Кабинет Школьного Психолога!\nДля продолжения выберите свою роль:",

reply\_markup=keyboard\_roles

)

def get\_role\_keyboard(role):

keyboard = types.ReplyKeyboardMarkup(resize\_keyboard=True)

options = {

"Students": ["Пройти психологический тест", "Анкеты от психолога", "Получить анонимную помощь", "Контент"],

"Psychologists": ["Опубликовать статью", "Ответить на анонимное сообщение", "Опубликовать анкету"],

"Admins": ["Заблокировать пользователя", "Разблокировать пользователя"]

}

keyboard.add(\*options.get(role, []))

return keyboard

@bot.message\_handler(func=lambda message: message.text in ["Я ученик", "Я психолог", "Я администратор"])

def handle\_role(message):

user\_states[message.chat.id] = message.text

bot.send\_message(message.chat.id, "Пожалуйста, введите ваше имя и фамилию через пробел.")

@bot.message\_handler(func=lambda message: isinstance(user\_states.get(message.chat.id), str))

def handle\_name(message):

try:

first\_name, last\_name = message.text.split()

except ValueError:

bot.send\_message(message.chat.id, "Пожалуйста, введите имя и фамилию через пробел.")

return

if not (first\_name.isalpha() and last\_name.isalpha()):

bot.send\_message(message.chat.id, "Имя и фамилия должны содержать только буквы. Пожалуйста, попробуйте снова.")

return

telegram\_id = str(message.chat.id)

role = user\_states[message.chat.id]

status, first\_name\_db, last\_name\_db = add\_user(first\_name, last\_name, telegram\_id, role)

welcome\_message = ""

if status == "exists":

welcome\_message = f"Добро пожаловать обратно, {first\_name\_db} {last\_name\_db}! Чем я могу помочь Вам сегодня?"

elif status == "added":

welcome\_message = f"Добро пожаловать, {first\_name} {last\_name}! Чем я могу помочь Вам сегодня?"

else:

bot.send\_message(message.chat.id, "Произошла ошибка при добавлении пользователя.")

return

user\_states.pop(message.chat.id)

bot.send\_message(message.chat.id, welcome\_message, reply\_markup=get\_role\_keyboard(role))

tests = {

"Тест на уровень стресса": [

"Часто ли вы чувствуете напряжение?",

"Трудно ли вам расслабиться даже после отдыха?",

"Бывают ли у вас головные боли из-за стресса?",

"Чувствуете ли вы себя подавленным?",

"Сложно ли вам сосредоточиться на задачах?"

],

"Тест на уровень тревожности": [

"Часто ли вы беспокоитесь о будущем?",

"Легко ли вас вывести из равновесия?",

"Чувствуете ли вы себя нервным в новых ситуациях?",

"Сложно ли вам расслабиться в общественных местах?",

"Часто ли вы испытываете физические симптомы тревоги (например, потливость)?"

],

"Тест на уровень депрессии": [

"Чувствуете ли вы себя подавленным большую часть времени?",

"Потеряли ли вы интерес к вещам, которые раньше приносили радость?",

"Чувствуете ли вы себя безнадежным?",

"Сложно ли вам вставать с постели по утрам?",

"Чувствуете ли вы себя уставшим даже после сна?"

]

}

@bot.message\_handler(func=lambda message: message.text == "Пройти психологический тест")

@check\_blocked

def choose\_test(message):

keyboard = types.ReplyKeyboardMarkup(resize\_keyboard=True)

keyboard.add(\*tests.keys())

bot.send\_message(message.chat.id, "Выберите тест:", reply\_markup=keyboard)

@bot.message\_handler(func=lambda message: message.text in tests)

def start\_test(message):

test\_name = message.text

questions = tests[test\_name]

answers = ["Нет", "Иногда", "Часто"]

user\_states[message.chat.id] = {"test\_name": test\_name, "question\_index": 0, "score": 0}

bot.send\_message(message.chat.id, questions[0], reply\_markup=types.ReplyKeyboardMarkup(resize\_keyboard=True).add(\*answers))

@bot.message\_handler(func=lambda message: message.chat.id in user\_states)

def handle\_test\_question(message):

user\_data = user\_states[message.chat.id]

test\_name = user\_data["test\_name"]

questions = tests[test\_name]

score\_increment = {"Нет": 1, "Иногда": 2, "Часто": 3}

user\_answer = message.text

if user\_answer in score\_increment:

user\_data["score"] += score\_increment[user\_answer]

user\_data["question\_index"] += 1

if user\_data["question\_index"] < len(questions):

bot.send\_message(message.chat.id, questions[user\_data["question\_index"]], reply\_markup=types.ReplyKeyboardMarkup(resize\_keyboard=True).add(\*score\_increment.keys()))

else:

score = user\_data["score"]

result = "Низкий" if score <= 10 else "Средний" if score <= 15 else "Высокий"

bot.send\_message(message.chat.id, f"Ваш уровень: {result}", reply\_markup=get\_role\_keyboard("Students"))

del user\_states[message.chat.id]

else:

bot.send\_message(message.chat.id, "Пожалуйста, выберите один из предложенных вариантов.")

@bot.message\_handler(func=lambda message: message.text in ["Анкеты от психолога", "Получить анонимную помощь", "Контент"])

@check\_blocked

def handle\_special\_requests(message):

if message.text == "Анкеты от психолога":

take\_questionnaire(message)

elif message.text == "Получить анонимную помощь":

anonymous\_help(message)

elif message.text == "Контент":

show\_content(message)

def take\_questionnaire(message):

with connection.cursor() as cursor:

cursor.execute("SELECT id, title FROM questionnaires")

questionnaires = cursor.fetchall()

if not questionnaires:

bot.send\_message(message.chat.id, "На данный момент нет доступных анкет!")

else:

response\_message = "Доступные анкеты:\n" + "\n".join(f"{q[0]}: {q[1]}" for q in questionnaires)

bot.send\_message(message.chat.id, response\_message + "Введите ID анкеты, чтобы начать.")

bot.register\_next\_step\_handler(message, process\_selected\_questionnaire)

def process\_selected\_questionnaire(message):

try:

questionnaire\_id = int(message.text)

except ValueError:

bot.send\_message(message.chat.id, "Пожалуйста, введите корректный ID анкеты.")

return

with connection.cursor() as cursor:

cursor.execute("SELECT id, question FROM questions WHERE questionnaire\_id = %s", (questionnaire\_id,))

questionnaire = cursor.fetchall()

if questionnaire:

user\_states[message.chat.id] = {

"questionnaire\_id": questionnaire\_id,

"current\_question\_index": 0,

"questionnaire": questionnaire

}

send\_question(message.chat.id, questionnaire)

else:

bot.send\_message(message.chat.id, "Анкета не найдена.")

def send\_question(chat\_id, questionnaire):

current\_index = user\_states[chat\_id]["current\_question\_index"]

if current\_index < len(questionnaire):

question = questionnaire[current\_index][1]

bot.send\_message(chat\_id, question)

bot.register\_next\_step\_handler\_by\_chat\_id(chat\_id, process\_answer)

else:

bot.send\_message(chat\_id, "Вы ответили на все вопросы анкеты.")

del user\_states[chat\_id]

def process\_answer(message):

chat\_id = message.chat.id

questionnaire\_id = user\_states[chat\_id]["questionnaire\_id"]

with connection.cursor() as cursor:

cursor.execute("SELECT StudentID FROM Students WHERE TelegramID = %s", (chat\_id,))

student\_id = cursor.fetchone()

if student\_id is None:

bot.send\_message(chat\_id, "Ошибка: пользователь не найден.")

return

cursor.execute("INSERT INTO answers (questionnaire\_id, student\_id, answer) VALUES (%s, %s, %s)",

(questionnaire\_id, student\_id[0], message.text))

connection.commit()

user\_states[chat\_id]["current\_question\_index"] += 1

send\_question(chat\_id, user\_states[chat\_id]["questionnaire"])

def anonymous\_help(message):

bot.send\_message(message.chat.id, "Пришлите анонимное сообщение, и психолог ответит на него.")

bot.register\_next\_step\_handler(message, handle\_anon\_response)

def handle\_anon\_response(message):

user\_id = message.chat.id

anon\_message\_text = message.text

try:

with connection.cursor() as cursor:

sqlInsertAnonMessage = "INSERT INTO AnonymousMessages (StudentID, MessageText) VALUES ((SELECT StudentID FROM Students WHERE TelegramID = %s), %s)"

cursor.execute(sqlInsertAnonMessage, (user\_id, anon\_message\_text))

connection.commit()

bot.send\_message(user\_id, "Ваше сообщение отправлено, психолог скоро на него ответит.")

except Exception as ex:

bot.send\_message(user\_id, f"Произошла ошибка: {ex}\nПопробуйте повторить позже!")

def show\_content(message):

with connection.cursor() as cursor:

cursor.execute("SELECT \* FROM Content")

content = cursor.fetchall()

if content:

response\_message = "Вот ваш контент на сегодня:\n" + "\n".join(f"ID: {item[0]}\nСодержание: {item[1]}" for item in content)

bot.send\_message(message.chat.id, response\_message)

else:

bot.send\_message(message.chat.id, "На сегодня контента нет.")

@bot.message\_handler(func=lambda message: message.text == 'Опубликовать статью')

def start\_article(message):

bot.send\_message(message.chat.id, "Пожалуйста, введите содержание статьи:")

bot.register\_next\_step\_handler(message, receive\_message)

def receive\_message(message):

article\_content = message.text

with connection.cursor() as cursor:

cursor.execute("INSERT INTO Content (ContentData) VALUES (%s)", (article\_content,))

connection.commit()

bot.send\_message(message.chat.id, "Статья успешно опубликована!")

# Блокировка и разблокировка пользователей

@bot.message\_handler(func=lambda message: message.text in ['Заблокировать пользователя', 'Разблокировать пользователя'])

def block\_or\_unblock\_user(message):

action = "блокировать" if message.text == 'Заблокировать пользователя' else "разблокировать"

bot.send\_message(message.chat.id, f"Введите ID пользователя для {action}:")

bot.register\_next\_step\_handler(message, lambda m: block\_user(m) if action == "блокировать" else unblock\_user(m))

def block\_user(message):

try:

idblockuser = int(message.text)

except ValueError:

bot.send\_message(message.chat.id, "Пожалуйста, введите ID пользователя корректно!")

return

with connection.cursor() as cursor:

cursor.execute("SELECT blockuser\_id FROM Block WHERE blockuser\_id = %s", (idblockuser,))

if cursor.fetchone() is not None:

bot.send\_message(message.chat.id, f"Пользователь с ID: {idblockuser} уже заблокирован!")

return

try:

cursor.execute("INSERT INTO Block (blockuser\_id) VALUES (%s)", (idblockuser,))

connection.commit()

bot.send\_message(message.chat.id, f"Пользователь с ID: {idblockuser} был заблокирован.")

except Exception as e:

bot.send\_message(message.chat.id, "Произошла ошибка при блокировке пользователя.")

def unblock\_user(message):

try:

unblockUserID = int(message.text)

except ValueError:

bot.send\_message(message.chat.id, "Пожалуйста, введите корректный ID пользователя!")

return

with connection.cursor() as cursor:

cursor.execute("DELETE FROM Block WHERE blockuser\_id = %s", (unblockUserID,))

connection.commit()

if cursor.rowcount > 0:

bot.send\_message(message.chat.id, f"Пользователь с ID: {unblockUserID} был разблокирован.")

else:

bot.send\_message(message.chat.id, f"Пользователь с ID: {unblockUserID} не найден в списке заблокированных!")

def is\_user\_blocked(user\_id):

with connection.cursor() as cursor:

cursor.execute("SELECT blockuser\_id FROM Block WHERE blockuser\_id = %s", (user\_id,))

return cursor.fetchone() is not None

@bot.message\_handler(func=lambda message: message.text == "Список заблокированных пользователей")

def block\_users(message):

with connection.cursor() as cursor:

cursor.execute("SELECT \* FROM Block")

block\_users = cursor.fetchall()

if block\_users:

response = "Заблокированные пользователи:\n" + "\n".join(f"ID: {user[0]}" for user in block\_users)

else:

response = "Нет заблокированных пользователей"

bot.send\_message(message.chat.id, response)

bot.infinity\_polling()

connection.close()